The Guido Engine
A toolbox for music scores rendering

C. Daudin, D. Fober, S. Letz and **Y. Orlarey**
Grame
Centre national de cration musicale
Lyon, France
{daudin, fober, letz, orlarey}@grame.fr

Abstract

The GUIDO Music Notation format (GMN) is a general purpose formal language for representing score level music in a platform independent plain text and human readable way. Based on this music representation format, the GUIDOLib provides a generic, portable library and API for the graphical rendering of musical scores. This paper gives an introduction to the music notation format and to the GUIDO graphic score rendering engine. An example of application, the *GuidoSceneComposer*, is next presented.

Keywords

GUIDO, Music notation, score layout.

1 Introduction

Computer music has started to investigate music score rendering very early [1][2][3]. Music notation codes such as DARMS [4], SCORE [5] or MuseData [6] have been designed to address representational issues and gave birth to associated programs (such as the SCORE Notation Program, widely used in engraving during the 1980s and 1990s) and a long history of derived or alternate formats.

Now, commercial music publishing software exist for more a decade and actually provide sophisticated but complex solutions for music engraving. Along these *closed* solutions, the toolbox approach has been investigated very early [7]. However, very few systems have reached maturity: the *Common Music Notation* system [8] could be considered as the best achievement. More recently, the *Expressive Notation Package* (ENP) [9] introduced another promising approach; both systems are Lisp based environments.

Another solution consists in designing compilers for producing music sheets from a textual music description. MusiXTEX is among these tools: it is a set of TEX macros to typeset music notation. Since MusiXTEX is powerful but hard to learn, preprocessors such as PMX and M-Tx have been designed to facilitate music input and layout. A more recent initiative is Lilypond [10], an open source software partially implemented in the language Scheme, its input music representation format is simple and intuitive, it includes automatic layout capabilities. Both systems produce PostScript, EPS or PDF files.

Based the GUIDO Music Notation format, the GUIDOLib project is a open source, cross-platform C/C++ library that provides score layout and rendering capabilities to its client applications. The music notation format is very close to the Lylipond format. The GUIDOLib mainly differs from the compilers approach in that it allows to embed music score rendering capabilities into standalone applications and to create scores dynamically.

This paper introduces first the GUIDO Music Notation format, next the GUIDO Engine and the GUIDO library API are presented. The last section presents the Qt support for GUIDO, along with a concrete example of GUIDO-Qt application, the *GuidoSceneComposer*.

2 The Guido Music Notation format

The GUIDO Music Notation format (GMN) [11] [12] has been designed by H. Hoos and K. Hamel more than ten years ago. It is a general purpose formal language for representing score level music in a platform independent plain text and human readable way. It is based on a conceptually simple but powerful formalism: its design concentrates on general musical concepts (as opposed to graphical features). A key feature of the GUIDO design is adequacy which means that simple musical concepts should be represented in a simple way and only complex notions should require complex representations.

2.1 Basic concepts

Basic GUIDO notation covers the representation of notes, rests, accidentals, single and multi-voiced music and the most common concepts from conventional music notation such as clefs, meter, key, slurs, ties, beaming, stem directions, etc. Notes are specified by their name (`a b c d e f g h`), optional accidentals ('#' and '&' for sharp and flat), an optional octave number and an optional duration.
Duration is specified in one of the forms:

```
'*'enum'/'denom dotting
'*'enum dotting
'/'denom dotting
```

where *enum* and *denom* are positive integers and *dotting* is either empty, '.', or '..', with the same semantic than the music notation. When *enum* or *denom* is omitted, it is assumed to be 1. The duration represents a whole note fractional.

When omitted, optional note description parts are assumed to be equal to the previous specification before in the current sequence.

Chords are described using comma separated notes enclosed in brackets e.g {c, e, g}

2.2 Guido tags

Tags are used to represent additional musical information, such as slurs, clefs, keys, etc. A basic tag has one of the forms:

```
\tagname
\tagname<param-list>
```

where `param-list` is a list of string or numerical arguments, separated by commas (','). In addition, a tag may have a time range and be applied to a series of notes (like slurs, ties etc.); the corresponding form is:

```
\tagname(note-series)
\tagname<param-list>(note-series)
```

The following GMN code illustrates the concision of the notation; figure 1 represents the corresponding GUIDO engine output.

```
[ \meter<"4/4"> \key<-2> c d e& f/8 g ]
```

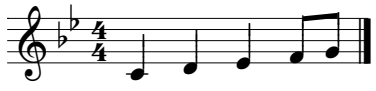

Figure 1: A simple GMN example

2.3 Notes sequences and segments

A note sequence is of the form `[tagged-notes]` where `tagged-notes` is a series of notes, tags, and tagged ranges separated by spaces. Note sequences represent single-voiced scores. Note segments represent multi-voiced scores; they are denoted by `{seq-list}` where `seq-list` is a list of note sequences separated by commas as shown by the example below:

```
{
 [\staff<1> \stemsUp \meter<"2/4">
  \beam(g2/32 e/16 c*3/32) c/8
  \beam(a1/16 c2 f)
  \beam(g/32 d/16 h1*3/32) d2/8
  \beam(h1/16 d2 g)],

 [\staff<1>\stemsDown g1/8 e
  \beam(g/16 d f a) a/8 e \beam(a/16 e g h)],

 [\staff<2> \stemsUp \meter<"2/4"> a0 f h c1],
 [\staff<2>\stemsDown f0 d g a]
}
```

The corresponding GUIDO engine output is given by figure 2.

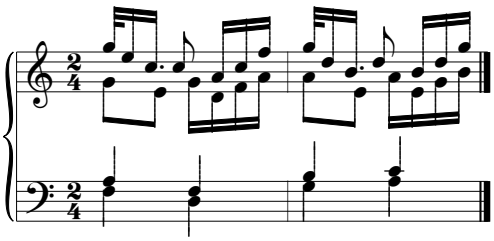

Figure 2: A multi-voices example

Additionally, the advanced GUIDO specification (not covered by this paper) provides exact formatting of the score.

3 The Guido Engine

Based on the GUIDO Music Notation format and initially designed by Kai Renz, the GUIDO Engine provides graphical rendering of musical scores, including automatic layout capabilities. At Grame's initiative, the engine has been reshaped under the form of a portable library and became an open source project in 2002, covered by the GNU LGPL license and hosted on SourceForge. Since 2002, the GUIDO engine has been maintained and extended by Grame.

The GUIDO Engine operates on a memory representation of the GMN format: the GUIDO Abstract Representation (GAR). This representation is transformed step by step to produce graphical score pages. Two kinds of processing are first applied to the GAR:

- GAR to GAR transformations which represents a logical layout transformation: part of the layout (such as beaming for example) may be computed from the GAR as well as expressed in GAR,
- the GAR is converted into a GUIDO Semantic Normal Form (GSNF). The GSNF is a

canonical form such that different semantically equivalent expressions have the same GSNF.

This GSNF is finally converted into a GUIDO Graphic Representation (GGR) that contains the necessary layout information and is directly used to draw the music score. This final step notably includes spacing and page breaking algorithms [13].

Note that although the GMN format allows for acurate music formatting using advanced GUIDO (see figure 3), the GUIDO Engine provides powerful automatic layout capabilities.

4 The Guido Library

The GUIDO Library is implemented is in C++ but the services of GUIDO Engine are available using a C API.

4.1 Score layout

The library provides functions to parse a GMN file and to create the corresponding GAR and GGR. GAR and GGR are referenced by opaque handles which are used as arguments of any function that operates on a score. For example: `GuidoParse (const char * filename)` provides conversion of a GMN file into a GGR handle returned as the function result. This handle may be next used to draw the score using the `GuidoOnDraw` function.

A typical code to draw a score from its GMN description is given below (see section 4.6 for `VGDevice` information):

```
void DrawGMNFile (char* filename, VGDevice* device)
{
  // data structure for engine initialization
  // uses fonts "guido2" and "times"
  GuidoInitDesc gd = { device, 0, "guido2", "times" };
  // Initialise the Guido Engine first
  GuidoInit (&gd);

  // declare a data structure for drawing
  GuidoOnDrawDesc desc;
  // and parse the GMN file to get a GGR handle
  // directly stored in the drawing struct
  desc.handle = GuidoParse (filename);

  // next setup the drawing parameters
  // (see the documentation for more details)
  desc.hdc = device;  // the output device
  desc.page = 1;      // the page to draw
  desc.updateRegion.erase = true;
  desc.scrollx = desc.scrolly = 0;
  desc.zoom = 1;      // no zoom
  desc.sizex = desc.sizey = 0;

  // and finally draws the score
  GuidoOnDraw (&desc);
}
```

4.2 Score pages access

The result of the score layout is a set of pages which size may be dynamically changed according to an application or a user needs. The library provides the necessary to change the page size, to query a score pages count, or the page number corresponding to a given music date. Note that only one page is drawn by the `GuidoOnDraw` function.

4.3 Engine settings

Score layout algorithms are controlled by a set of parameters which are global to the GUIDO engine. The library provides an API to query and modify these parameters. It includes optimal page fill control, springs and space force control, systems distance and systems distribution.

4.4 The Guido Factory

The GUIDO Engine may be feeded with computer generated music using the GUIDO Factory. The GUIDO Factory API provides a set of functions to create a GAR from scratch and to convert it into a GGR. The GUIDO Factory is a state machine that operates on implicit current elements: for example, once you open a voice (`GuidoFactoryOpenVoice()`), it becomes the current voice and all subsequent created events are implicitly added to this current voice.

The GUIDO Factory state includes the current score, voice, chord, note (or rest) and tag. Some elements of the factory state reflects the GUIDO formal specification; unless otherwise specified, new notes will implicitly carry the current duration and octave.

A music score dynamic construction is very close to the textual GUIDO description: the Factory API handles GAR objects that have a one to one relationship with the notation format. Once the score has been dynamically built, a call to `GuidoFactoryCloseMusic()` returns a GUIDO handle to a GAR, directly usable with `GuidoFactoryMakeGR()`, which returns a GUIDO handle to a GGR, directly usable with the main services of the library. Logical layout is performed before returning the GAR handle and graphical layout is performed before returning the GGR handle.

4.5 Graphic Mappings

Along with the GGR, the GUIDO Engine maintains a tree of graphical elements for each page of the score, as illustrated by figure 4. Each element has a bounding box and a date. Positions

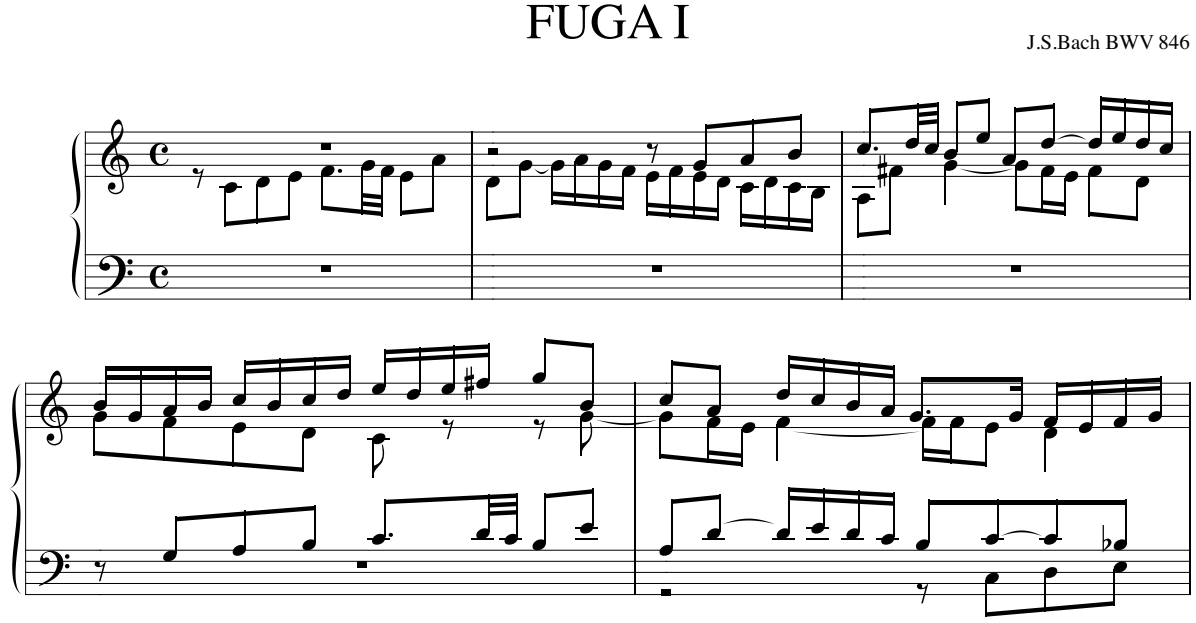

Figure 3: A complex layout example

of the elements are stored as pixel coordinates. Top left corner of the score is at position (0, 0).

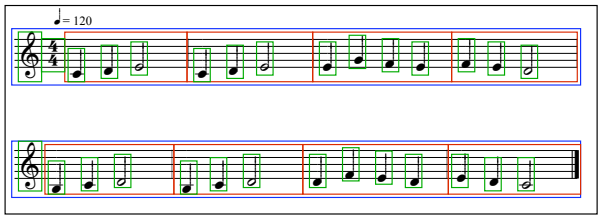

Figure 4: Bounding rects of a score elements

The GUIDO library provides a specific API to query the score map and to retrieve elements by type, position and date.

4.6 The Virtual Graphic System

The virtual graphic system is intended as an abstract layer covering platform dependencies at graphic level. It represents a set of abstract classes adressing the basic needs of an application: printing text, drawing on the screen or offscreen, etc. The set of abstract classes includes:

- a `VGDevice` class: specialized on drawing onscreen or offscreen
- a `VGFont` class: to cover fonts management
- a `VGSystem` class: to cover allocation of specific `VGDevice` and `VGFont` objects.

This set of classes is implemented for different target platforms: support is provided for GDI (Windows), Quartz (Mac OS X), GTK (GNU Linux), OpenGL and more recently for Qt.

5 Guido Qt support

Qt is a cross-platform application development framework [14], widely used for the development of GUI programs[1].

5.1 GuidoQt classes

A set of classes has been developed to use the GUIDO library with Qt; they are organized in 3 layers (figure 5):

- *low level*: GDeviceQt, GFontQt & GSystemQt: Qt implementation of the Virtual Graphic System
- *middle level*: QGuidoPainter: a class that uses GDeviceQt, GFontQt & GSystemQt, and offers a higher-level interface
- *high level*: ready-to-use Qt classes that uses the QGuidoPainter to parse and draw GUIDO scores:
  - QGuidoWidget, a QWidget
  - QGuidoSPageItem & QGuidoMPageItem, two QGraphicsItem displaying, respectively, one score's page at a time, and all the pages
  - Guido2Image, to export GUIDO scores to various image formats

[1] see: http://www.qtsoftware.com

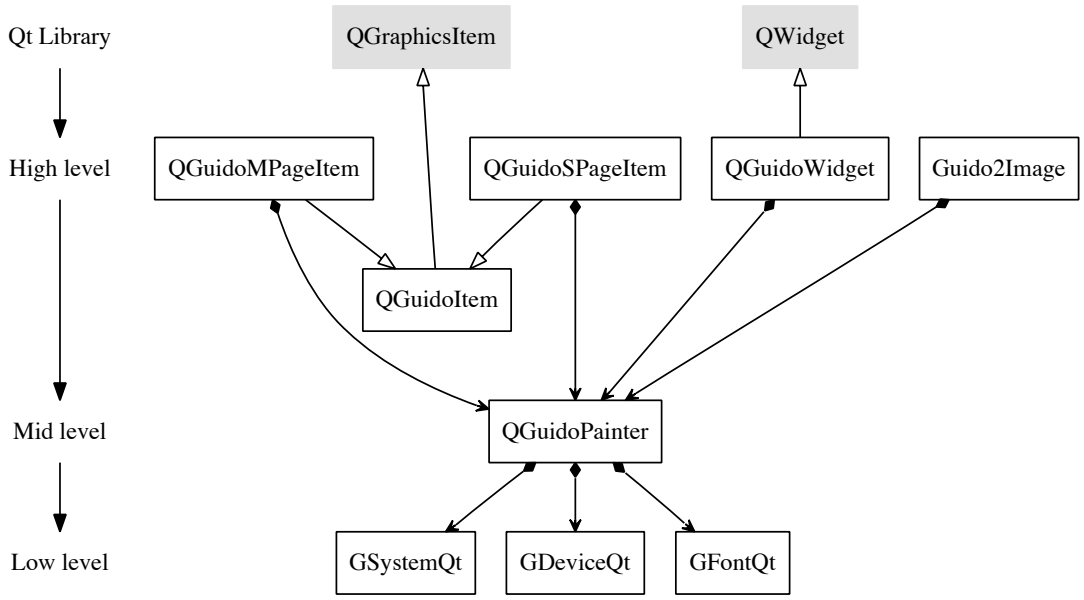

Figure 5: GuidoQt class diagram

Here is a short example using the QGuidoPainter that shows how to export a GUIDO score to an image:

```
#include <QApplication>
#include <QPainter>
#include <QImage>
#include "QGuidoPainter.h"
int main(int argc , char* argv[])
{
   QApplication app( argc , argv );

   //Guido Engine initialization
   QGuidoPainter::startGuidoEngine();

   //Create a QGuidoPainter...
   QGuidoPainter* p = QGuidoPainter::create();
   //...and give it some Guido Music Notation
   p->setGMNCode( "[c/8 d e g e d c]" );

   //Get the size of the score's first and only page
   int pageIndex = 1;
   QSizeF s = p->pageSizeMM( pageIndex );

   //Create a blank image, using the size of the score
   QImage image(s.toSize()*10 , QImage::Format_ARGB32);
   image.fill( QColor(Qt::white).rgb() );

   //Draw the score with the QGuidoPainter, via QPainter.
   QPainter painter( &image );
   p->draw( &painter , pageIndex , image.rect() );

   QGuidoPainter::destroyGuidoPainter( p );

   //Destroy the Guido Engine resources
   QGuidoPainter::stopGuidoEngine();

   image.save( "myScore.png" );
   return 0;
}
```

And here is another example showing how to display a GUIDO Score using a QGuidoWidget:

```
#include <QApplication>
#include "QGuidoWidget.h"
#include "QGuidoPainter.h"
int main(int argc, char *argv[])
{
   QApplication app(argc, argv);

   // Guido Engine initialization
   QGuidoPainter::startGuidoEngine();

   // Create a QGuidoWidget...
   QGuidoWidget w;
   //...and give it some Guido Music Notation
   w.setGMNCode( "[e d c]" );
   w.show();

   // That's it !
   int result = app.exec();

   // Destroy the Guido Engine resources
   QGuidoPainter::stopGuidoEngine();
   return result;
}
```

5.2 The Guido Scene Composer

The *GuidoSceneComposer* is a graphic IDE for GUIDO, allowing to manipulate GUIDO scores in a graphic scene.

5.2.1 A Guido Music Notation learning tool

The *GuidoSceneComposer* is an effective environment to learn the GMN:

- when typing GMN, the corresponding score is instantly updated, making it very simple to try different notations;
- a help palette (figure 6) lists a large assortment of GUIDO expressions and tags,

so that no external documentation is necessary;

- the GMN text editor uses a synthax highlighter, for a clearer GMN reading;
- import MusicXML and MuseData (limited support) scores;
- export the GUIDO scores to a pdf or an image.

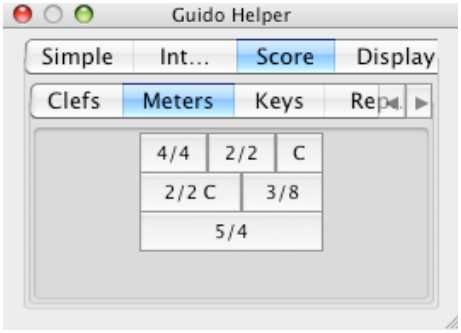

Figure 6: The GUIDO Help palette

5.2.2 A score graphic composer

The *GuidoSceneComposer* allows to you to compose graphic scenes with different scores:

- move, resize, copy & paste the scores, to create a graphically complex musical scene,
- import various formats of pictures to enrich the scene,
- add textual annotations,
- export the scene to an image or a pdf.

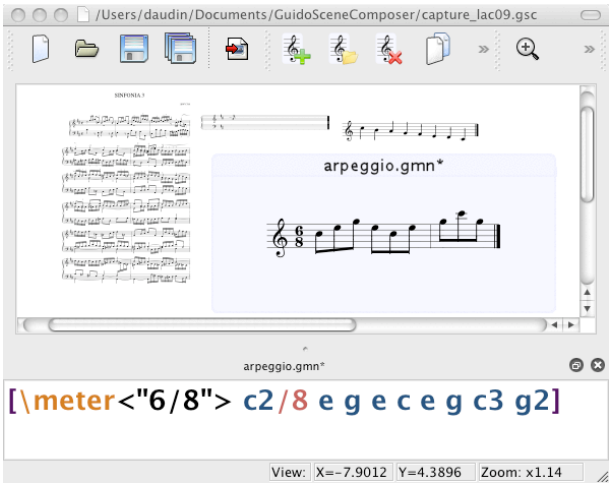

Figure 7: The GUIDO Scene Composer

6 Conclusions

Based on the GUIDO Music Notation format, the GUIDOLib is a unique open source, platform-independant library, for embedding score rendering into a standalone application. The recent support of the widely-used cross-platform Qt library makes it even more accessible to software developers.

In the future, we plan to extend the graphical possibilities of the *GuidoSceneComposer* giving to users more drawing tools; it would then become an interesting application to create extended modern music scores. Support of import/export in other common music score formats is also among our concerns.

Finally, we plan to add score composition features, like putting scores in sequence or in parallel, cutting the head, the tail, the top or the bottom voices of a score, transposition, or duration change, using an homogeneous approach, where scores are both the target and the argument of the operations.

The GUIDO library is available on Sourceforge at `http://guidolib.sourceforge.net`

References

[1] David A. Gomberg. A Computer-Oriented System for Music Printing. In *Computers and the Humanities*, volume 11, pages 63–80. Pergamon Press, 1977.

[2] Donald Alvin Bird. *Music Notation by Computer*. PhD thesis, Indiana University, 1984.

[3] John S. Gourlay, A. Parrish, D. Roush, F. Sola, and Y. Tien. Computer Formatting of Music. Technical report OSU-CISRC-2/87-TR3, Department of Computer and Information Science, The Ohio State University, 1987.

[4] E. Selfridge-Field. DARMS, Its Dialiects, and Its Uses. In *Beyond MIDI, The handbook of Musical Codes.*, pages 163–174. MIT Press, 1997.

[5] Smith Leland. SCORE. In *Beyond MIDI, The handbook of Musical Codes.*, pages 252–280. MIT Press, 1997.

[6] Walter B. Hewlett. MuseData: Multipurpose Representation. In Selfridge-Field E., editor, *Beyond MIDI, The handbook of Musical Codes.*, pages 402–447. MIT Press, 1997.

[7] Assayag G. and D. Timis. A ToolBox for Music Notation. In *Proceedings of the International Computer Music Conference*, pages 173–178, 1986.

[8] Bill Schottstaedt. Common Music Notation. In Selfridge-Field E., editor, *Beyond MIDI, The handbook of Musical Codes.*, pages 217–222. MIT Press, 1997.

[9] Mika Kuuskankare and Michael Laurson. ENP - Music Notation Library based on Common Lisp and CLOS. In *Proceedings of the International Computer Music Conference*, pages 131–134. ICMA, 2001.

[10] Han-Wen Nienhuys and Jan Nieuwenhuizen. LilyPond, a system for automated music engraving. In *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003)*, May 2003.

[11] Hoos H., Hamel K. A., Renz K., and Kilian J. The GUIDO Music Notation Format - a Novel Approach for Adequately Representing Score-level Music. In *Proceedings of the International Computer Music Conference*, pages 451–454. ICMA, 1998.

[12] H. H. Hoos and K. A. Hamel. The GUIDO Music Notation Format Specification - version 1.0, part 1: Basic GUIDO. Technical report TI 20/97, Technische Universitat Darmstadt, 1997.

[13] Kai Renz. *Algorithms and Data Structures for a Music Notation System based on GUIDO Music Notation.* PhD thesis, Technischen Universität Darmstadt, 2002.

[14] Jasmin Blanchette and Mark Summerfield. *C++ GUI Programming with Qt 4 (2nd Edition).* Prentice Hall, 2008.